17 | API 文档：如何生成 Swagger API 文档 ？

2021-07-03 孔令飞

《Go 语言项目开发实战》 课程介绍 >

讲述：孔令飞
时长 10:03 大小 9.22M

你好，我是孔令飞。

作为一名开发者，我们通常讨厌编写文档，因为这是一件重复和缺乏乐趣的事情。但是在开发过程中，又有一些文档是我们必须要编写的，比如 API 文档。

一个企业级的 Go 后端项目，通常也会有个配套的前端。为了加快研发进度，通常是后端和前端并行开发，这就需要后端开发者在开发后端代码之前，先设计好 API 接口，提供给前端。所以在设计阶段，我们就需要生成 API 接口文档。

一个好的 API 文档，可以减少用户上手的复杂度，也意味着更容易留住用户。好的 API 文档也可以减少沟通成本，帮助开发者更好地理解 API 的调用方式，从而节省时间，提高开发效率。这时候，我们一定希望有一个工具能够帮我们自动生成 API 文档，解放我们的双

手。Swagger 就是这么一个工具，可以帮助我们**生成易于共享且具有足够描述性的 API 文档**。

接下来，我们就来看下，如何使用 Swagger 生成 API 文档。

Swagger 介绍

Swagger 是一套围绕 OpenAPI 规范构建的开源工具，可以设计、构建、编写和使用 REST API。Swagger 包含很多工具，其中主要的 Swagger 工具包括：

Swagger 编辑器：基于浏览器的编辑器，可以在其中编写 OpenAPI 规范，并实时预览 API 文档。https://editor.swagger.io 就是一个 Swagger 编辑器，你可以尝试在其中编辑和预览 API 文档。

Swagger UI：将 OpenAPI 规范呈现为交互式 API 文档，并可以在浏览器中尝试 API 调用。

Swagger Codegen：根据 OpenAPI 规范，生成服务器存根和客户端代码库，目前已涵盖了 40 多种语言。

Swagger 和 OpenAPI 的区别

我们在谈到 Swagger 时，也经常会谈到 OpenAPI。那么二者有什么区别呢？

OpenAPI 是一个 API 规范，它的前身叫 Swagger 规范，通过定义一种用来描述 API 格式或 API 定义的语言，来规范 RESTful 服务开发过程，目前最新的 OpenAPI 规范是 OpenAPI 3.0（也就是 Swagger 2.0 规范）。

OpenAPI 规范规定了一个 API 必须包含的基本信息，这些信息包括：

对 API 的描述，介绍 API 可以实现的功能。

每个 API 上可用的路径（/users）和操作（GET /users，POST /users）。

每个 API 的输入 / 返回的参数。

验证方法。

联系信息、许可证、使用条款和其他信息。

所以，你可以简单地这么理解：OpenAPI 是一个 API 规范，Swagger 则是实现规范的工具。

另外，要编写 Swagger 文档，首先要会使用 Swagger 文档编写语法，因为语法比较多，这里就不多介绍了，你可以参考 Swagger 官方提供的OpenAPI Specification来学习。

用 go-swagger 来生成 Swagger API 文档

在 Go 项目开发中，我们可以通过下面两种方法来生成 Swagger API 文档：

第一，如果你熟悉 Swagger 语法的话，可以直接编写 JSON/YAML 格式的 Swagger 文档。建议选择 YAML 格式，因为它比 JSON 格式更简洁直观。

第二，通过工具生成 Swagger 文档，目前可以通过swag和go-swagger两个工具来生成。

对比这两种方法，直接编写 Swagger 文档，不比编写 Markdown 格式的 API 文档工作量小，我觉得不符合程序员“偷懒”的习惯。所以，本专栏我们就使用 go-swagger 工具，基于代码注释来自动生成 Swagger 文档。为什么选 go-swagger 呢？有这么几个原因：

> go-swagger 比 swag 功能更强大：go-swagger 提供了更灵活、更多的功能来描述我们的 API。
>
> 使我们的代码更易读：如果使用 swag，我们每一个 API 都需要有一个冗长的注释，有时候代码注释比代码还要长，但是通过 go-swagger 我们可以将代码和注释分开编写，一方面可以使我们的代码保持简洁，清晰易读，另一方面我们可以在另外一个包中，统一管理这些 Swagger API 文档定义。
>
> 更好的社区支持：go-swagger 目前有非常多的 Github star 数，出现 Bug 的概率很小，并且处在一个频繁更新的活跃状态。

你已经知道了，go-swagger 是一个功能强大的、高性能的、可以根据代码注释生成 Swagger API 文档的工具。除此之外，go-swagger 还有很多其他特性：

> 根据 Swagger 定义文件生成服务端代码。
>
> 根据 Swagger 定义文件生成客户端代码。

校验 Swagger 定义文件是否正确。

启动一个 HTTP 服务器，使我们可以通过浏览器访问 API 文档。

根据 Swagger 文档定义的参数生成 Go model 结构体定义。

可以看到，使用 go-swagger 生成 Swagger 文档，可以帮助我们减少编写文档的时间，提高开发效率，并能保证文档的及时性和准确性。

这里需要注意，如果我们要对外提供 API 的 Go SDK，可以考虑使用 go-swagger 来生成客户端代码。但是我觉得 go-swagger 生成的服务端代码不够优雅，所以建议你自行编写服务端代码。

目前，有很多知名公司和组织的项目都使用了 go-swagger，例如 Moby、CoreOS、Kubernetes、Cilium 等。

安装 Swagger 工具

go-swagger 通过 swagger 命令行工具来完成其功能，swagger 安装方法如下：

复制代码

```
$ go get -u github.com/go-swagger/go-swagger/cmd/swagger
$ swagger version
dev
```

swagger 命令行工具介绍

swagger 命令格式为`swagger [OPTIONS] <command>`。可以通过`swagger -h`查看 swagger 使用帮助。swagger 提供的子命令及功能见下表：

子命令	功能
diff	对比两个swagger文档的差异
expand	展开Swagger定义文档中的$ref
flatten	展平swagger文档
generate	生成swagger文档、客户端代码、服务端代码等
ini	初始化一个swagger定义文档，初始化时可以指定一些配置
mix	合并Swagger文档
serv	启动HTTP服务，以查看Swagger文档
validate	验证Swagger定义文件是否正确
version	打印swagger命令版本

如何使用 swagger 命令生成 Swagger 文档？

go-swagger 通过解析源码中的注释来生成 Swagger 文档，go-swagger 的详细注释语法可参考官方文档。常用的有如下几类注释语法：

注释语法	功能
swagger:meta	定义API接口全局基本信息
swagger:route	定义路由信息
swagger:parameters	定义API请求参数
swagger:response	定义API响应参数
swagger:model	定义可以复用的Go数据结构
swagger:allOf	嵌入其他Go结构体
swagger:strfmt	定义格式化的字符串
swagger:ignore	定义需要忽略的结构体

解析注释生成 Swagger 文档

swagger generate 命令会找到 main 函数，然后遍历所有源码文件，解析源码中与 Swagger 相关的注释，然后自动生成 swagger.json/swagger.yaml 文件。

这一过程的示例代码为 gopractise-demo/swagger。目录下有一个 main.go 文件，定义了如下 API 接口：

复制代码

```
1 package main
2
```

```go
import (
    "fmt"
    "log"
    "net/http"

    "github.com/gin-gonic/gin"

    "github.com/marmotedu/gopractise-demo/swagger/api"
    // This line is necessary for go-swagger to find your docs!
    _ "github.com/marmotedu/gopractise-demo/swagger/docs"
)

var users []*api.User

func main() {
    r := gin.Default()
    r.POST("/users", Create)
    r.GET("/users/:name", Get)

    log.Fatal(r.Run(":5555"))
}

// Create create a user in memory.
func Create(c *gin.Context) {
    var user api.User
    if err := c.ShouldBindJSON(&user); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"message": err.Error(), "code": 10
        return
    }

    for _, u := range users {
        if u.Name == user.Name {
            c.JSON(http.StatusBadRequest, gin.H{"message": fmt.Sprintf("user %
            return
        }
    }

    users = append(users, &user)
    c.JSON(http.StatusOK, user)
}

// Get return the detail information for a user.
func Get(c *gin.Context) {
    username := c.Param("name")
    for _, u := range users {
        if u.Name == username {
            c.JSON(http.StatusOK, u)
            return
        }
    }

    c.JSON(http.StatusBadRequest, gin.H{"message": fmt.Sprintf("user %s not ex
```

```
55 }
```

main 包中引入的 **User struct** 位于 gopractise-demo/swagger/api 目录下的 user.go 文件：

复制代码

```
1  // Package api defines the user model.
2  package api
3
4  // User represents body of User request and response.
5  type User struct {
6      // User's name.
7      // Required: true
8      Name string `json:"name"`
9
10     // User's nickname.
11     // Required: true
12     Nickname string `json:"nickname"`
13
14     // User's address.
15     Address string `json:"address"`
16
17     // User's email.
18     Email string `json:"email"`
19 }
```

`// Required: true`说明字段是必须的，生成 Swagger 文档时，也会在文档中声明该字段是必须字段。

为了使代码保持简洁，我们在另外一个 Go 包中编写带 go-swagger 注释的 API 文档。假设该 Go 包名字为 docs，在开始编写 Go API 注释之前，需要在 main.go 文件中导入 docs 包：

复制代码

```
1  _ "github.com/marmotedu/gopractise-demo/swagger/docs"
```

通过导入 docs 包，可以使 go-swagger 在递归解析 main 包的依赖包时，找到 docs 包，并解析包中的注释。

在 gopractise-demo/swagger 目录下，创建 docs 文件夹：

复制代码

```
$ mkdir docs
$ cd docs
```

在 docs 目录下，创建一个 doc.go 文件，在该文件中提供 API 接口的基本信息：

复制代码

```
// Package docs awesome.
//
// Documentation of our awesome API.
//
//     Schemes: http, https
//     BasePath: /
//     Version: 0.1.0
//     Host: some-url.com
//
//     Consumes:
//     - application/json
//
//     Produces:
//     - application/json
//
//     Security:
//     - basic
//
//    SecurityDefinitions:
//    basic:
//      type: basic
//
// swagger:meta
package docs
```

Package docs 后面的字符串 awesome 代表我们的 HTTP 服务名。`Documentation of our awesome API`是我们 API 的描述。其他都是 go-swagger 可识别的注释，代表一定的意义。最后以`swagger:meta`注释结束。

编写完 doc.go 文件后，进入 gopractise-demo/swagger 目录，执行如下命令，生成 Swagger API 文档，并启动 HTTP 服务，在浏览器查看 Swagger：

```
$ swagger generate spec -o swagger.yaml
$ swagger serve --no-open -F=swagger --port 36666 swagger.yaml

2020/10/20 23:16:47 serving docs at http://localhost:36666/docs
```

复制代码

-o：指定要输出的文件名。swagger 会根据文件名后缀.yaml 或者.json，决定生成的文件格式为 YAML 或 JSON。

--no-open：因为是在 Linux 服务器下执行命令，没有安装浏览器，所以使 --no-open 禁止调用浏览器打开 URL。

-F：指定文档的风格，可选 swagger 和 redoc。我选用了 redoc，因为觉得 redoc 格式更加易读和清晰。

--port：指定启动的 HTTP 服务监听端口。

打开浏览器，访问 http://localhost:36666/docs ，如下图所示：

如果我们想要 JSON 格式的 Swagger 文档，可执行如下命令，将生成的 swagger.yaml 转换为 swagger.json：

```
$ swagger generate spec -i ./swagger.yaml -o ./swagger.json
```

复制代码

接下来，我们就可以编写 API 接口的定义文件（位于 gopractise-demo/swagger/docs/user.go 文件中）：

```
package docs
```

复制代码

```
import (
    "github.com/marmotedu/gopractise-demo/swagger/api"
)

// swagger:route POST /users user createUserRequest
// Create a user in memory.
// responses:
//   200: createUserResponse
//   default: errResponse

// swagger:route GET /users/{name} user getUserRequest
// Get a user from memory.
// responses:
//   200: getUserResponse
//   default: errResponse

// swagger:parameters createUserRequest
type userParamsWrapper struct {
    // This text will appear as description of your request body.
    // in:body
    Body api.User
}

// This text will appear as description of your request url path.
// swagger:parameters getUserRequest
type getUserParamsWrapper struct {
    // in:path
    Name string `json:"name"`
}

// This text will appear as description of your response body.
// swagger:response createUserResponse
type createUserResponseWrapper struct {
    // in:body
    Body api.User
}

// This text will appear as description of your response body.
// swagger:response getUserResponse
type getUserResponseWrapper struct {
    // in:body
    Body api.User
}

// This text will appear as description of your error response body.
// swagger:response errResponse
type errResponseWrapper struct {
    // Error code.
    Code int `json:"code"`

    // Error message.
```

```
54      Message string `json:"message"`
55  }
```

user.go 文件说明：

swagger:route：`swagger:route`代表 API 接口描述的开始，后面的字符串格式为`HTTP方法 URL Tag ID`。可以填写多个 tag，相同 tag 的 API 接口在 Swagger 文档中会被分为一组。ID 是一个标识符，`swagger:parameters`是具有相同 ID 的`swagger:route`的请求参数。`swagger:route`下面的一行是该 API 接口的描述，需要以英文点号为结尾。`responses:`定义了 API 接口的返回参数，例如当 HTTP 状态码是 200 时，返回 createUserResponse，createUserResponse 会跟`swagger:response`进行匹配，匹配成功的`swagger:response`就是该 API 接口返回 200 状态码时的返回。

swagger:response：`swagger:response`定义了 API 接口的返回，例如 getUserResponseWrapper，关于名字，我们可以根据需要自由命名，并不会带来任何不同。getUserResponseWrapper 中有一个 Body 字段，其注释为`// in:body`，说明该参数是在 HTTP Body 中返回。`swagger:response`之上的注释会被解析为返回参数的描述。api.User 自动被 go-swagger 解析为`Example Value`和`Model`。我们不用再去编写重复的返回字段，只需要引用已有的 Go 结构体即可，这也是通过工具生成 Swagger 文档的魅力所在。

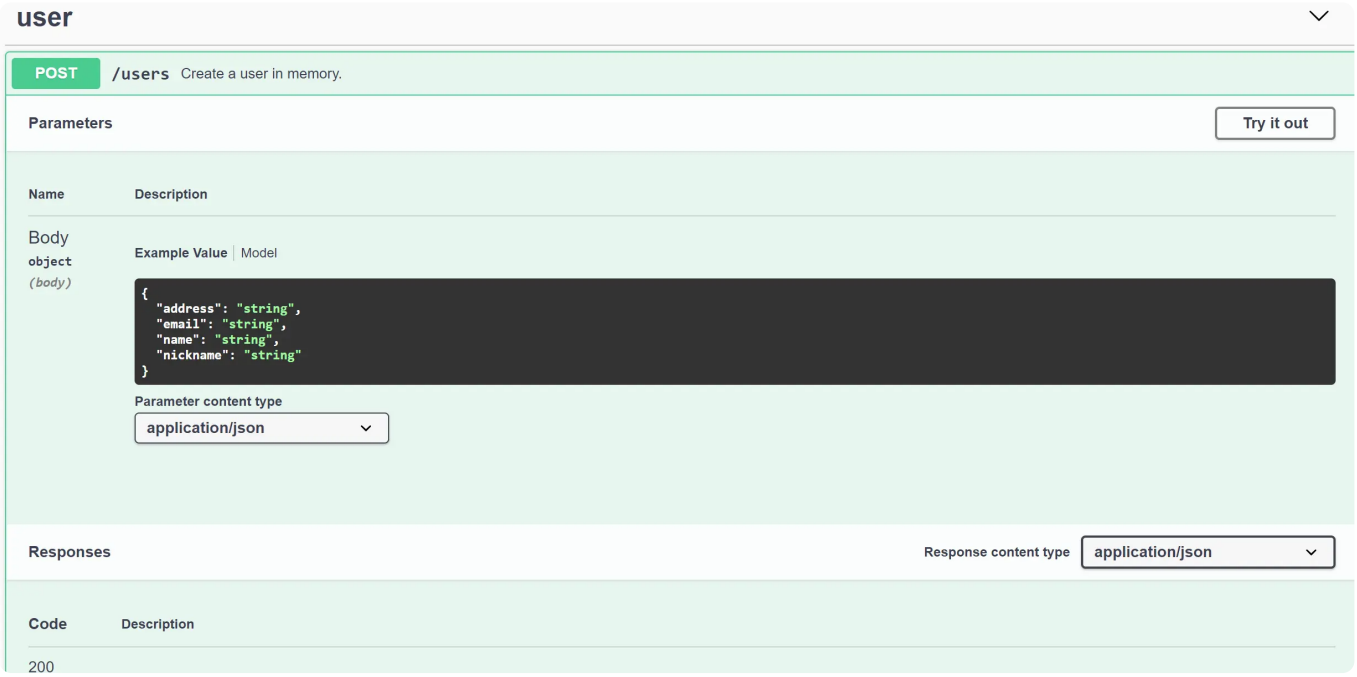
swagger:parameters：`swagger:parameters`定义了 API 接口的请求参数，例如 userParamsWrapper。userParamsWrapper 之上的注释会被解析为请求参数的描述，`// in:body`代表该参数是位于 HTTP Body 中。同样，userParamsWrapper 结构体名我们也可以随意命名，不会带来任何不同。`swagger:parameters`之后的 createUserRequest 会跟`swagger:route`的 ID 进行匹配，匹配成功则说明是该 ID 所在 API 接口的请求参数。

进入 gopractise-demo/swagger 目录，执行如下命令，生成 Swagger API 文档，并启动 HTTP 服务，在浏览器查看 Swagger：

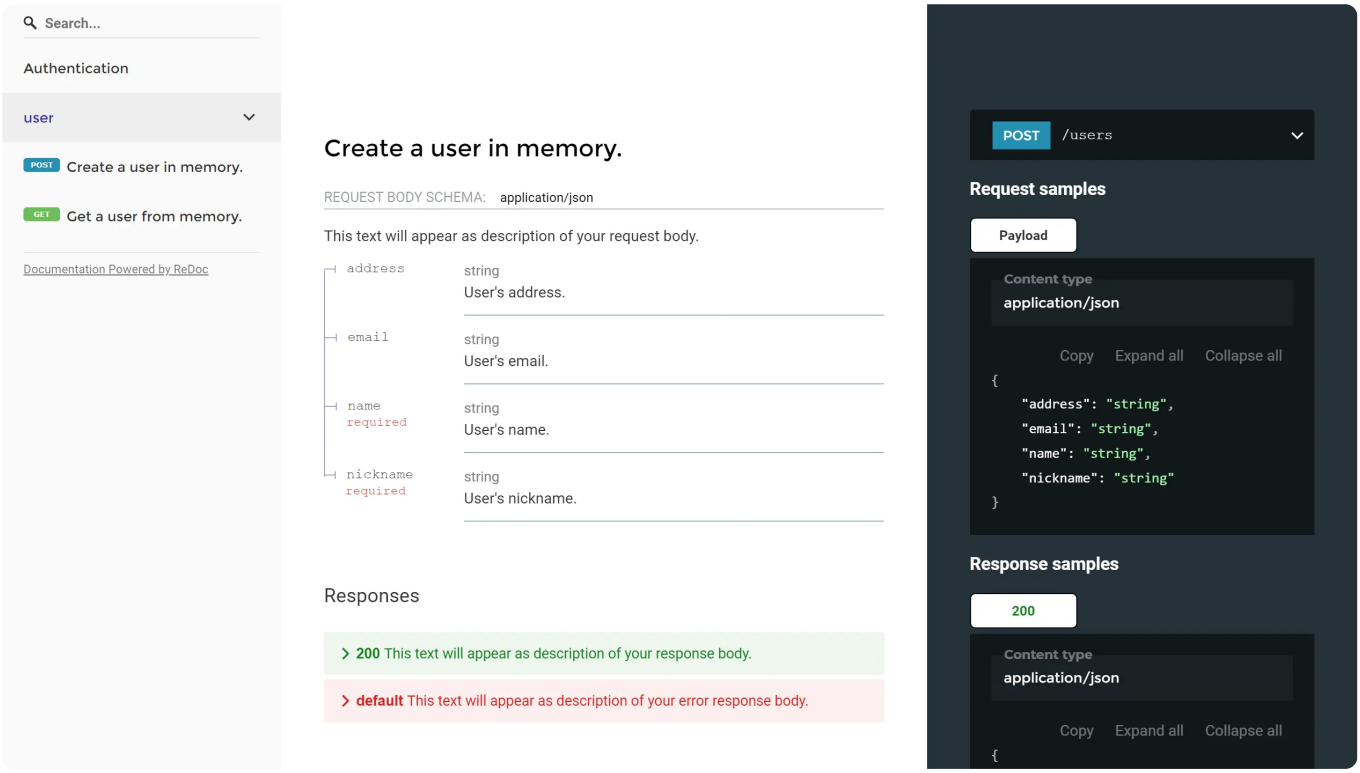
复制代码

```
1 $ swagger generate spec -o swagger.yaml
2 $ swagger serve --no-open -F=swagger --port 36666 swagger.yaml
3 2020/10/20 23:28:30 serving docs at http://localhost:36666/docs
```

打开浏览器，访问 http://localhost:36666/docs ，如下图所示：



上面我们生成了 swagger 风格的 UI 界面，我们也可以使用 redoc 风格的 UI 界面，如下图所示：



go-swagger 其他常用功能介绍

上面，我介绍了 swagger 最常用的 generate、serve 命令，关于 swagger 其他有用的命令，这里也简单介绍一下。

1. 对比 Swagger 文档

复制代码

```
$ swagger diff -d change.log swagger.new.yaml swagger.old.yaml
$ cat change.log

BREAKING CHANGES:
=================
/users:post Request - Body.Body.nickname.address.email.name.Body : User - Dele
compatibility test FAILED: 1 breaking changes detected
```

2. 生成服务端代码

我们也可以先定义 Swagger 接口文档，再用 swagger 命令，基于 Swagger 接口文档生成服务端代码。假设我们的应用名为 go-user，进入 gopractise-demo/swagger 目录，创建 go-user 目录，并生成服务端代码：

复制代码

```
$ mkdir go-user
$ cd go-user
$ swagger generate server -f ../swagger.yaml -A go-user
```

上述命令会在当前目录生成 cmd、restapi、models 文件夹，可执行如下命令查看 server 组件启动方式：

复制代码

```
$ go run cmd/go-user-server/main.go -h
```

3. 生成客户端代码

在 go-user 目录下执行如下命令：

复制代码

```
$ swagger generate client -f ../swagger.yaml -A go-user
```

上述命令会在当前目录生成 client，包含了 API 接口的调用函数，也就是 API 接口的 Go SDK。

4. 验证 Swagger 文档是否合法

```
$ swagger validate swagger.yaml
2020/10/21 09:53:18
The swagger spec at "swagger.yaml" is valid against swagger specification 2.0
```

5. 合并 Swagger 文档

```
$ swagger mixin swagger_part1.yaml swagger_part2.yaml
```

IAM Swagger 文档

IAM 的 Swagger 文档定义在iam/api/swagger/docs目录下，遵循 go-swagger 规范进行定义。

iam/api/swagger/docs/doc.go文件定义了更多 Swagger 文档的基本信息，比如开源协议、联系方式、安全认证等。

更详细的定义，你可以直接查看 iam/api/swagger/docs 目录下的 Go 源码文件。

为了便于生成文档和启动 HTTP 服务查看 Swagger 文档，该操作被放在 Makefile 中执行（位于iam/scripts/make-rules/swagger.mk文件中）：

```
.PHONY: swagger.run
swagger.run: tools.verify.swagger
  @echo "===========> Generating swagger API docs"
  @swagger generate spec --scan-models -w $(ROOT_DIR)/cmd/genswaggertypedocs -

.PHONY: swagger.serve
swagger.serve: tools.verify.swagger
  @swagger serve -F=redoc --no-open --port 36666 $(ROOT_DIR)/api/swagger/swagg
```

Makefile 文件说明：

tools.verify.swagger：检查 Linux 系统是否安装了 go-swagger 的命令行工具 swagger，如果没有安装则运行 go get 安装。

swagger.run：运行 `swagger generate spec` 命令生成 Swagger 文档 swagger.yaml，运行前会检查 swagger 是否安装。`--scan-models` 指定生成的文档中包含带有 swagger:model 注释的 Go Models。`-w` 指定 swagger 命令运行的目录。

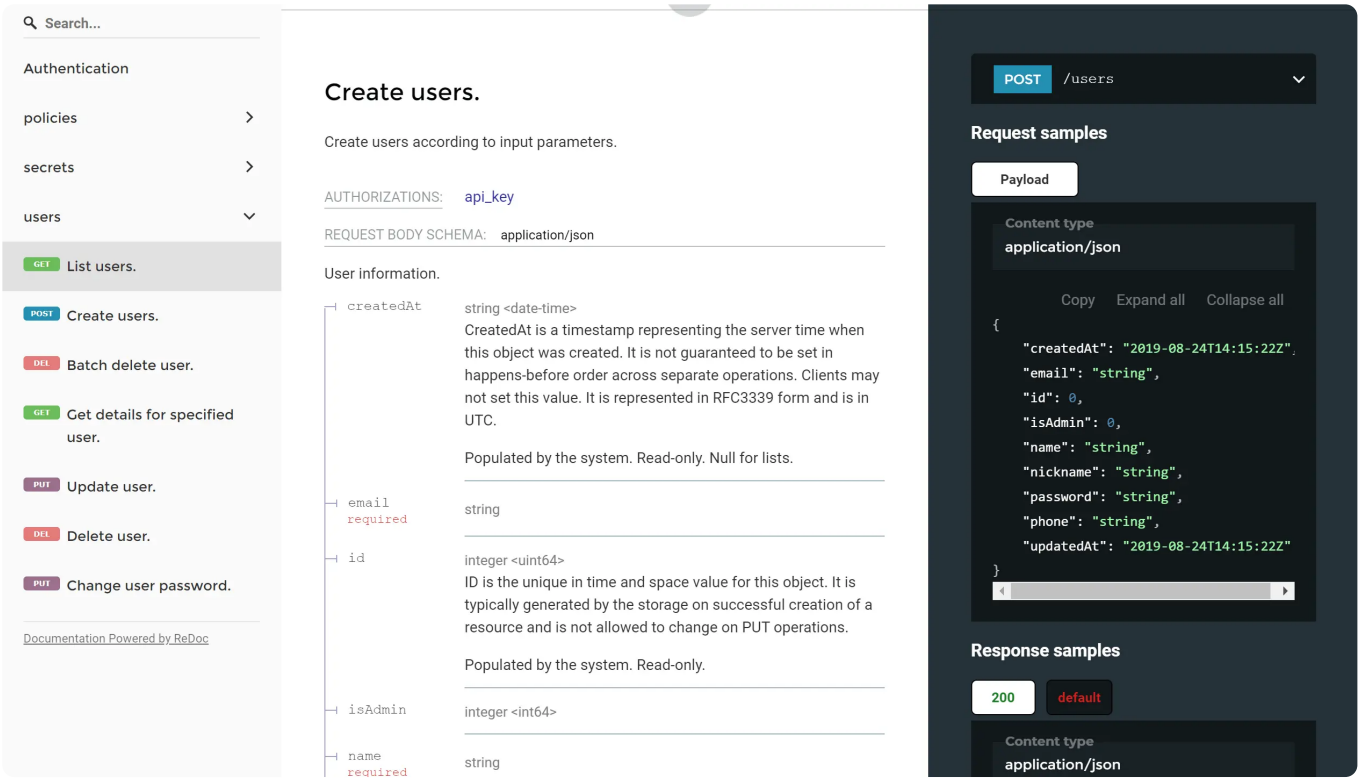
swagger.serve：运行 `swagger serve` 命令生成 Swagger 文档 swagger.yaml，运行前会检查 swagger 是否安装。

在 iam 源码根目录下执行如下命令，即可生成并启动 HTTP 服务查看 Swagger 文档：

复制代码

```
$ make swagger
$ make serve-swagger
2020/10/21 06:45:03 serving docs at http://localhost:36666/docs
```

打开浏览器，打开`http://x.x.x.x:36666/docs`查看 Swagger 文档，x.x.x.x 是服务器的 IP 地址，如下图所示：



IAM 的 Swagger 文档，还可以通过在 iam 源码根目录下执行`go generate ./...`命令生成，为此，我们需要在 iam/cmd/genswaggertypedocs/swagger_type_docs.go 文件中，添加`//go:generate`注释。如下图所示：

```
5  package main
6
7  //go:generate swagger generate spec -o ../../api/swagger/swagger.yaml --scan-models
8
9  import (
10     _ "github.com/marmotedu/iam/api/swagger/docs"
11 )
12
13 func main() {
14 }
```

总结

在做 Go 服务开发时，我们要向前端或用户提供 API 文档，手动编写 API 文档工作量大，也难以维护。所以，现在很多项目都是自动生成 Swagger 格式的 API 文档。提到 Swagger，很多开发者不清楚其和 OpenAPI 的区别，所以我也给你总结了：OpenAPI 是一个 API 规范，Swagger 则是实现规范的工具。

在 Go 中，用得最多的是利用 go-swagger 来生成 Swagger 格式的 API 文档。go-swagger 包含了很多语法，我们可以访问 Swagger 2.0进行学习。学习完 Swagger 2.0 的语法之后，就可以编写 swagger 注释了，之后可以通过

```
$ swagger generate spec -o swagger.yaml
```

来生成 swagger 文档 swagger.yaml。通过

```
$ swagger serve --no-open -F=swagger --port 36666 swagger.yaml
```

来提供一个前端界面，供我们访问 swagger 文档。

为了方便管理，我们可以将 `swagger generate spec` 和 `swagger serve` 命令加入到 Makefile 文件中，通过 Makefile 来生成 Swagger 文档，并提供给前端界面。

课后练习

1. 尝试将你当前项目的一个 API 接口，用 go-swagger 生成 swagger 格式的 API 文档，如果中间遇到问题，欢迎在留言区与我讨论。
2. 思考下，为什么 IAM 项目的 swagger 定义文档会放在 iam/api/swagger/docs 目录下，这样做有什么好处？

欢迎你在留言区与我交流讨论，我们下一讲见。

分享给需要的人，Ta订阅后你可得 24 元现金奖励

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 16 | 代码检查：如何进行静态代码检查？

下一篇 18 | 错误处理（上）：如何设计一套科学的错误码？

精选留言 (5)

写留言

daz2yy
2021-07-05

一点想法，在小团队里，其实也没那么规范，能简单就简单点，目前用的是 Postman 生成的文档，考虑的点：
1. 一方面程序员自己调试接口必然会用到接口调用工具，postman大家也比较熟悉；
2. 另一方面开发完之后，它能直接生成在线文档，不用重新去定义文档使用的 API 的字段、结构体等，感觉会方便很多，而且能实时同步修改（接口有变动可以同步反映到文...
展开

作者回复: 文档怎么编写，这里其实没有一个统一的、标准的规范，因为不同团队对API文档的要求、对使用工具的选择都不一样。

这一讲介绍的方法，仅供参考。具体使用哪种，你们可以参考这一讲介绍的方法， 再结合自己的实际需求和情况，进行调研之后，再做选择。

如果你们觉得postman、或者一些在线工具能满足你们的需求，其实我觉得可能要不用swagger更好些，毕竟可视化、而且真的很方便、效率很高。

感谢反馈。

4

helloworld

2021-07-03

思考题可能的好处：1. 方便和项目根路径下的帮助文档目录docs做区分；2. 路径层次清晰辨认功能；3. 方便启api文档的http服务？

展开

作者回复: 说的很好。再补充下：放在api目录下，说明这个是api的定义文件。API文档聚合在一个目录下，后期维护，查看都很方便

1

2021-07-09

老师厉害 把软件工程方方面面通过一个项目讲了
相信看完这个再去看大型开源项目 比如kubernates应该会容易很多
聚焦在论文核心实现部分就可以
知道一个大项目是怎么造出来的 包括什么 再去学自然容易多了

展开

遇见@z

2021-07-08

看着好麻烦，我们公司自己写的框架通过扫描ast语法树生成openapi，代码即文档

作者回复: 强，这个没有固定的方法，可以根据需要自己选择。这是个好方法，后面我看下怎么补下这种方式

宙斯

2021-07-06

文档版本比较多时（有的接口有3-4个版本），放在一个swagger文档下是否还需要加版本以区分呢？

作者回复: 需要的